

Net-Centric Implementation Framework

Part 1: Overview

Part 2: ASD(NII) Checklist Guidance

Part 3: Migration Guidance

Part 4: Node Guidance

Part 5: Developer Guidance

**Part 6: Contracting Guidance for
Acquisition**

V 2.0

30 April 2007



Net-Centric Enterprise Solutions for Interoperability (NESI) is a collaborative activity of the USN Program Executive Office for Command, Control, Communications, Computers and Intelligence (PEO C4I); the USAF Electronic Systems Center (ESC); and the Defense Information Systems Agency (DISA).

Approved for public release; distribution is unlimited.

Table of Contents

1 NESI Implementation.....	1
1.1 References	1
1.2 Overview	2
1.3 Releasability statement	3
1.4 Vendor Neutrality	3
1.5 Disclaimer	3
1.6 Contributions and Comments	3
1.7 Collaboration Site	3
2 Introduction.....	4
2.1 Audience	4
3 Migrating to a Net-Centric Warfare Environment	5
3.1 ASD(NII)/DoD CIO Categories	5
3.1.1 Non-Compliant (Retire).....	6
3.1.2 Legacy Being Sustained (Retain).....	6
3.1.3 Legacy Being Transformed (Refresh)	6
3.1.4 New Start / In Development.....	6
3.2 NESI Migration Levels	6
4 Selecting a Migration Level	8
4.1 Assessing Risk	8
4.1.1 Client and Presentation Tiers	9
4.1.2 Middle Tier	9
4.1.3 Data Tier	9
4.1.4 Multi-User Applications	9
4.1.5 Cross-Domain Security (CDS).....	9
4.2 Assessing Scope	10
4.3 Level 1: Minimum Upgrade.....	11
4.3.1 Overarching	11
4.3.2 Client and Presentation Tier	12
4.3.3 Middle Tier	13
4.3.4 Data Tier	14
4.4 Level 2: Mid-Level Upgrade.....	14
4.4.1 Overarching	15
4.4.2 Client and Presentation Tier	16
4.4.3 Middle Tier	16
4.4.4 Data Tier	17
4.5 Level 3: Net-Centric Upgrade	18
4.5.1 Overarching	19
4.5.2 Client and Presentation Tier	19
4.5.3 Middle Tier	20
4.5.4 Data Tier	20
4.6 Level 4: Full Net-Centric Integration	20
4.6.1 Overarching	21
5 Migrating COE Systems and Applications.....	23
5.1 Selecting an Approach	23
5.2 Analyzing COE Capabilities.....	23
5.3 Decision Tree.....	24
5.4 Examples of Mixed COE/Non-COE Systems.....	25
5.5 Migrating Systems with Basic COE Dependency.....	25
5.6 Migrating Systems with COE Component Dependencies	26
5.6.1 COE Alerts Dependency	26
5.6.2 COE APM/CDS Dependency	26

5.6.3	COE JMTK Dependency	26
5.6.4	COE CMP Dependency.....	26
5.6.5	COE ICSF Dependency	26
5.7	Migrating COE Components Using a Bridge Approach	27
6	Mapping Maintenance Actions to Enterprise Technology Objectives.....	29

(This page is intentionally blank.)

1 NESI Implementation

NESI Part 3: Migration Guidance is the third of six parts of the NESI Net-Centric Implementation Document Set. Part 3 guidance is intended for the program managers and DoD contractors of existing programs. These programs use pre-planned product improvement or maintenance funds to incorporate net-centric characteristics. This guidance presents a strategy to migrate deployed applications to the net-centric paradigm during the maintenance phase. It describes how to implement a phased software migration strategy to deliver net-centric capability and to fulfill current contractual and program maintenance obligations.

Section 1 of Part 3 contains brief NESI background information. For more introductory information, see the first part of this document set, *NESI Part 1: Overview*.

Note: Part 3 is undergoing a comprehensive revision intended for the next release of the NESI Implementation document set.

1.1 References

- (a) DoD Directive 5000.1, *The Defense Acquisition System*, 24 November 2003.
- (b) DoD Instruction 5000.2, *Operation of the Defense Acquisition System*, 12 May 2003.
- (c) DoD Directive 8100.1, *Global Information Grid (GIG) Overarching Policy*, 21 November 2003.
- (d) DoD Directive 4630.5, *Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 05 May 2004.
- (e) DoD Instruction 4630.8, *Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 30 June 2004.
- (f) DoD Directive 5101.7, *DoD Executive Agent for Information Technology Standards*, 21 May 2004.
- (g) *DoD Global Information Grid (GIG) Architecture, Version 2.0*, August 2003.
- (h) *DoD Architecture Framework (DoDAF), Version 1.0*, 9 February 2004.
- (i) *DoD Net-Centric Data Strategy*, DoD Chief Information Officer, 9 May 2003.
- (j) CJCSI 3170.01E, *Joint Capabilities Integration and Development System*, 11 May 2005.
- (k) CJCSM 3170.01B, *Operation of the Joint Capabilities Integration and Development System*, 11 May 2005.
- (l) CJCSI 6212.01D, *Interoperability and Supportability of Information Technology and National Security Systems*, 8 March 2006.
- (m) *Net-Centric Operations and Warfare Reference Model (NCOW RM)*, Version 1.1 (Draft), 8 November 2004.
- (n) *Net-Centric Checklist, V2.1.3*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 12 May 2004.

- (o) *A Modular Open Systems Approach (MOSA) to Acquisition, Version 2.0*, September 2004.
- (p) *DoD IT Standards Registry (DISR)*, <http://disronline.disa.mil>.
- (q) *Net-Centric Attributes List*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, June 2004.
- (r) *Global Information Grid (GIG) Key Interface Profiles (KIPs) Framework (DRAFT)*, Version 0.95, 7 October 2005.

1.2 Overview

Net-Centric Enterprise Solutions for Interoperability (NESI) provides, for all phases of the acquisition of net-centric solutions, actionable guidance that meets DoD Network-Centric Warfare goals. The guidance in NESI is derived from the higher level, more abstract concepts provided in various directives, policies and mandates such as the *Net-Centric Operations and Warfare Reference Model (NCOW RM)* and the ASD(NII) *Net-Centric Checklist*, references (m) and (n), respectively. As currently structured, NESI guidance is captured in documents covering architecture, design and implementation; a compliance checklist; and a collaboration environment that includes a repository of guidance statements and code examples.

More specifically, NESI is a body of architectural and engineering knowledge that guides the design, implementation, maintenance, evolution, and use of the Information Technology (IT) portion of net-centric solutions for military application. NESI provides specific technical recommendations that a DoD organization can use as references. Stated another way, NESI can be thought of as a reference set of compliant instantiations of these directives.

NESI is derived from both a studied examination of enterprise-level needs and more importantly from the collective practical experience of recent and on-going program-level implementations. It is based on today's technologies and probable near-term technology developments. It describes the practical experience of system developers within the context of a minimal top-down technical framework. Most, if not all, of the guidance in NESI is in line with commercial best practice in the area of enterprise computing.

NESI applies to all phases of the acquisition process as defined in references (a) and (b) and applies to both new and legacy programs. NESI provides explicit counsel for building in net-centricity from the ground up and for migrating legacy systems to greater degrees of net-centricity.

NESI subsumes a number of references and directives; in particular, the Air Force *C2 Enterprise Technical Reference Architecture (C2ERA)*¹ and the Navy *Reusable Applications Integration and Development Standards (RAPIDS)*.² Initial authority for NESI is per the Memorandum of Agreement between Commander, Space and Naval Warfare Systems Command (SPAWAR), Navy PEO C4I & Space and the United States Air Force Electronic Systems Center, dated 22 December 2003, Subject: Cooperation Agreement for Net-Centric Solutions for Interoperability

¹ Air Force C2 Enterprise Technical Reference Architecture, v3.0-14, 1 December 2003.

² RAPIDS Reusable Application Integration and Development Standards, Navy PEO C4I & Space, December 2003 (DRAFT V1.5).

(NESI). The Defense Information Systems Agency (DISA) formally joined the NESI effort in 2006.

1.3 Releasability statement

This document has been cleared for public release by competent authority in accordance with DoD Directive 5230.9 and is granted *Distribution Statement A: Approved for public release; distribution is unlimited*. Obtain electronic copies of this document at <http://nesipublic.spawar.navy.mil>.

1.4 Vendor Neutrality

The NESI documentation sometimes refers to specific vendors and their products in the context of examples and lists. However, NESI is vendor-neutral. Mentioning a vendor or product is not intended as an endorsement, nor is a lack of mention intended as a lack of endorsement.

Code examples typically use open-source products since NESI is built on the open-source philosophy. NESI accepts inputs from multiple sources so the examples tend to reflect whatever tools the contributor was using or knew best. However, the products described are not necessarily the best choice for every circumstance. Users are encouraged to analyze specific project requirements and choose tools accordingly. There is no need to obtain, or ask contractors to obtain, the open-source tools that appear as examples in this guide. Similarly, any lists of products or vendors are intended only as references or starting points, and not as a list of recommended or mandated options.

1.5 Disclaimer

Every effort has been made to make NESI documentation as complete and accurate as possible. Even with frequent updates, this documentation may not always immediately reflect the latest technology or guidance. Also, references and links to external material are as accurate as possible; however, they are subject to change or may have additional access requirements such as Public Key Infrastructure (PKI), Common Access Card (CAC) use, and user accounts.

1.6 Contributions and Comments

NESI is an open-source project that will involve the entire development community. Anyone is welcome to contribute comments, corrections, or relevant knowledge to the guides via the Change Request tab on the NESI Public site, <http://nesipublic.spawar.navy.mil>, or via the following email address: nesi@spawar.navy.mil.

1.7 Collaboration Site

The Navy has established a collaboration site to support NESI community interaction. It is located at <https://nesi.spawar.navy.mil> (user registration required). Use this site for collaborative software development across distributed teams.

2 Introduction

Moving to a net-centric environment is a high priority of DoD leadership. NESI is taking a lead role in executing that vision. However, there are few or no additional dollars available for net-centricity. This requires using current resources more effectively. To transition successfully to a net-centric environment, programs need guidance that provides clear objectives and suggests improvements that can occur in conjunction with routine maintenance activity.

NESI advocates an incremental migration strategy to move applications towards this goal. Programs and contracts should use existing maintenance dollars to migrate applications to a system capable of Network Centric Warfare (NCW) while meeting current maintenance obligations. This approach leverages the DoD investment in deployed systems and training.

NESI Part 3: Migration Guidance (this document) contains the following guidance sections:

- Section 3, *Migrating to a Net-Centric Warfare Environment*: defines incremental migration strategies tailored to the type of application.
- Section 4, *Selecting a Migration Level*: provides tools for assessing the risk and scope of migration, identifies three levels of upgrade, then discusses how to implement upgrades in each tier.
- Section 5, *Migrating COE Systems and Applications*: recommends approaches to migrating COE-based systems to a net-centric infrastructure.
- Section 6, *Mapping Maintenance Actions to Enterprise Technology Objectives*: maps the maintenance actions described in Section 4 to the net-centric operational attributes described in *NESI Part 1: Overview*, such as capability on demand.

2.1 Audience

The intended audience for this document includes the following:

- Program managers
- Deputy program managers
- Contracting officers
- Chief engineers
- Contractor personnel

3 Migrating to a Net-Centric Warfare Environment

The technical migration strategy outlined in this document is built around a planned migration of functionality out of the current stovepipe systems and Common Operating Environment (COE) segments into a set of componentized applications, N-tier layered systems, separate node-based infrastructures, and new services. This migration strategy is based on fiscal limitations and the need to continue supporting current applications.³

Application functionality will be duplicated or wrapped, not necessarily removed, from current systems until all clients are using services instead of legacy stovepipe systems. Not all upgrades need to occur at the same time. Developers should identify and refactor application logic during planned maintenance activities. When application code is updated in a specific area, take the opportunity to add the appropriate net-centric environment upgrades.

This concurrent strategy entails insulating the structure of each system or application to enable proceeding independently of other enterprise systems and applications. *NESI Part 5: Developer Guidance* discusses insulation techniques such as multi-tier architectures, connectors, wrappers, adaptors, facades, proxies, bridges, and abstract interface classes.

In many cases, incremental migration is more efficient and carries less risk than a direct cutover. The benefits include the following:

- Staying within the current acquisition frameworks
- Leveraging common development opportunities
- Providing reasonable incentive for participation
- Offering short-term, tangible results that do not disrupt ongoing software development efforts
- Reducing cost as opposed to an expensive stop and re-engineer strategy

The main risk of this approach is that people may try to apply a one-size-fits-all strategy to all programs. This section discusses how to mitigate this risk by tailoring the migration to different types of programs.⁴ To that end, NESI defines “migration levels” that identify different levels of adaptation and maps them to the NESI strategic goals.

3.1 ASD(NII)/DoD CIO Categories

Reference (n) assigns programs to net-centric categories. The following sections identify each category and list relevant documentation.

³ This initial policy document focuses on software-related systems. More detailed network and transport level guidance will follow.

⁴ Real-time and closed-loop systems will implement a separate set of guidance (e.g., Open Architecture) that may not share all of the net-centric attributes.

The effort required to implement net-centricity in an application varies based on the target application. The spectrum ranges from already Web-enabled, multi-tiered, service-based applications to single-tiered, proprietary, closed stovepipes. In some cases, developers may wrap the entire application into one or several high-level coarse-grained interfaces. Then, the system components can migrate during subsequent iterations.

3.1.1 Non-Compliant (Retire)

Programs that do not exhibit net-centric capabilities and are not essential for continued operations or business processes will be guided towards termination.

3.1.2 Legacy Being Sustained (Retain)

Current programs that do not yet exhibit net-centric capabilities and are not planned for transformation, but are essential for current operations. If minimal cost growth is obtainable, programs in sustainment mode should attempt to meet the minimum criteria. *This document applies to some programs in this category, on a case-by-case basis.*

3.1.3 Legacy Being Transformed (Refresh)

Current programs that have an established plan to comply with net-centric capabilities and DoD domain requirements are in this group. *This document specifically addresses programs in this category.*

The matrix in Section 6, *Mapping Maintenance Actions to Enterprise Technology Objectives*, explains how to make net-centric enhancements that are of the same scope as typical maintenance or fix actions. The objective is to stretch existing maintenance dollars to build net-centric capabilities as well. If more money is available, programs can achieve higher objectives.

3.1.4 New Start / In Development

Programs that are born net-centric are in this group. They meet NCW requirements and are fully compliant with DoD net-centric models.

3.2 NESI Migration Levels

The table below illustrates the relationship of the ASD(NII)/DoD CIO categories to the NESI technical migration levels.

Table 1 – Correspondence between Checklist and NESI Levels

ASD(NII)/DoD CIO Checklist Categories	NESI Migration Levels
Non-Compliant (Retire)	Level 0 (As-Is) – Point to Point Legacy Interfaces
Legacy Being Sustained (Retain)	Level 1 – Migration to N-tier Structure
	Level 2 – Access to Legacy Data and Applications
Legacy Being Transformed (Refresh)	Level 3 – Legacy Applications Transformed
New Start/In Development	Level 4 – Fully Integrated Applications and Databases

The following figure depicts these levels of migration. This document discusses the technical activities involved in migrating to Levels 1, 2, and 3. *NESI Part 2: ASD(NII) Checklist Guidance* presents the criteria for Level 4.

Provide Flexibility through Multiple Levels of Migration

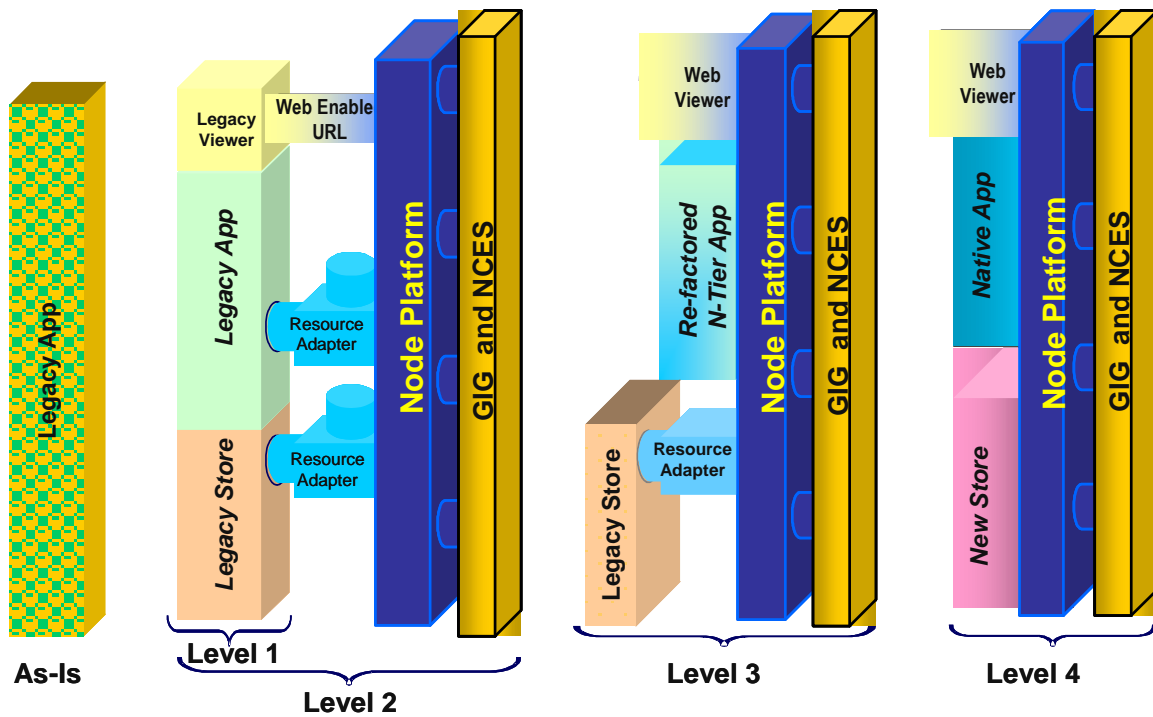


Figure 1 – Levels of Migration

4 Selecting a Migration Level

This section identifies several application migration levels. Each level lists technical net-centric upgrade actions for each of the operational attributes listed in *NESI Part 1: Net-Centric Overview*. These upgrades enable programs to implement the technical and application attributes they need to become net-centric and qualify for the ASD(NII)/DoD CIO “Refresh” category.

As one finds when buying a car, packaged options are more economical than random selections of options. We believe that this will also be true for systems migrating to net-centricity. The levels discussed here organize upgrades into logical groups and consider effort, complexity, and cost in a consistent manner.

NESI identifies four major levels of upgrades that range from minimal changes to a full net-centric integration. Program managers select the appropriate level based on risk and resource costs. See Sections 4.1, *Assessing Risk* and 4.2, *Assessing Scope*, before identifying a migration level..

- **Level 1, Minimum Upgrade.** Factor the application into tiers, modularize application code into components, and create public Application Programming Interfaces (APIs).
- **Level 2, Mid-Level Upgrade.** Using adapters, connect the application through its public APIs to the Node Platform Infrastructure (NPI).
- **Level 3, Net-Centric Upgrade.** Migrate application code to the Node Platform Infrastructure and prepare the application for the net-centric environment (namespaces, metadata, XML).
- **Level 4, Full Net-Centric Integration.** Enable legacy applications to function more fully in the net-centric environment. These enhancements are typically more expensive and time-consuming.

The levels are not compliance levels, and programs will not be judged against them. Moreover, program managers are free to select upgrades from multiple levels to suit the needs of their applications and the availability of resources.

It is essential to select upgrades that support structural insulation. This approach allows each modification to proceed independently. Carefully consider the best approach to applications that cannot be modified during other maintenance activities.

4.1 Assessing Risk

The NESI technical approach is based on migrating applications to an N-tier architecture. The following list describes some risks associated with net-centric upgrades. Consider these when choosing the upgrade level. Applications that migrate from an environment with a single or limited number of users, or that migrate into a cross-domain security environment, require additional evaluation.

4.1.1 Client and Presentation Tiers

These upgrades carry the lowest risk and the highest return, because they make the application more net-centric without affecting the operation of the application.

Depending on the legacy application architecture, it may be more appropriate to merge the client and presentation tiers. This guidance applies in both cases.

4.1.2 Middle Tier

These upgrades entail higher risk for applications that are not component-based or not structured with well-defined interfaces. Mitigate the risk by targeting smaller areas of the middle tier rather than the full application.

4.1.3 Data Tier

These upgrades also entail higher risk. The risk is especially high for applications that lack well-defined interfaces, are not insulated from the database or data stream, or store all the business logic in the database. Mitigate risk by using a combination of approaches such as switching to Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC), removing detailed business logic and algorithms from the data tier, and targeting small areas rather than reworking the entire application at once.

There are multiple technologies for exposing data to the enterprise. Providing a data-tier-service API involves significant costs and risks, which careful analysis and design can mitigate. Make sure to address data modeling, API construction, data element granularity, and XML format conversion. For example, share information between nodes via middle-tier services and not via SQL over the network.

As an example, for database applications, do not start by pulling all the business logic out of the database. Instead, leave the stored procedures in place and migrate to an open-standards abstraction layer such as ODBC/JDBC. In subsequent iterations, separate detailed business logic and algorithms from the database. Database Management System (DBMS) stored procedures, triggers and constraint checks are the optimal approach for inserting data, manipulating data, cascading deletions, enforcing constraints, and referential integrity.

For data stream applications, try implementing an open-standards wrapper abstraction layer to cover a subset of protocols. Extend the abstraction layer in subsequent iterations.

4.1.4 Multi-User Applications

These upgrades entail significant risk. Scaling a single-user application to be net-accessible and consequently multi-user is quite complex and requires careful planning. Some of the issues that need to be addressed are concurrency, locking, priority, transactions, state transitions, failover, security, and logging.

4.1.5 Cross-Domain Security (CDS)

Adopting CDS is often the highest risk area, since CDS designs and implementation are aimed at complex, multinational information sharing. Future development efforts should determine whether to incorporate a CDS design or use enterprise CDS services. Since it is very difficult to introduce CDS into a system during the maintenance phase, the CDS options have been moved

to beyond Level 3. See *NESI Part 2: ASD(NII) Checklist Guidance* for instructions on new development.

Current projects may be able to make use of a trusted CDS data service during maintenance upgrades, depending on the architecture. This could provide users with a single common data source and could enable other applications running at different security levels to use the data.

For projects that are looking at major restructuring, use CDS enterprise services as they are developed and deployed.

4.2 Assessing Scope

The Scope Assessment Matrix, below, correlates maintenance actions with net-centric upgrade actions. The matrix provides program managers with a rule-of-thumb guide for selecting net-centric upgrades based on the resources available for maintenance actions.

The matrix groups maintenance actions by resources available. It groups net-centric upgrades into the levels discussed above, based on risk and resource costs. For example, applications that only have the resources to perform fixes to Software Trouble Reports (STRs) would generally focus on Level 1 net-centric upgrades, achieving minimal net-centricity.

Table 2 – Scope Assessment Matrix

Maintenance Actions	Resources Available	Level 1, Minimum Upgrade	Level 2, Mid-Level Upgrade	Level 3, Net-Centric Upgrade
Software Trouble Reports (STRs) only	\$	X		
New functionality and STRs	\$\$	X	X	X (partial)
Application restructuring; major upgrades	\$\$\$	X	X	X

These development effort levels do not address DoD PKI Policy requirements.⁵ Individual programs are responsible for that compliance.

Early net-centric upgrades will probably implement upgrade options from multiple levels, depending on the degree of net-centricity, robustness, and maturity of an application. In general, though, applications will start with a basic set of upgrades and evolve toward a full-featured, net-centric environment.⁶

Once STRs and Change Packages (CP) are prioritized from Configuration Control Board (CCB) reviews, use the table above and the maintenance options listed in the levels below to determine the effect of the maintenance actions proposed for the upgrade cycle. Once determined, use the

⁵ See Department of Defense Instruction 8520.2, *Public Key Infrastructure (PKI) and Public Key (PK) Enabling*, 1 April 2004.

⁶ It is not necessary to complete all upgrades in one level before developing against subsequent levels. The state of an application will determine which upgrades can be done, in what order, and from which category.

matrix in Section 6, *Mapping Maintenance Actions to Enterprise Technology Objectives*, to assess the net-centric enabling accomplishments for this upgrade cycle.

4.3 Level 1: Minimum Upgrade

This level contains the lower-cost, basic upgrades that enable the application to participate in a net-centric environment. The theme for Level 1 is to prepare the application for migration to the net-centric environment. It begins by factoring the application into tiers and insulating it from the enterprise. As part of this preparation, Level 1 recommends a number of self-assessments for specific technical issues. In addition, performing various configuration and provisioning changes make the application easier to deploy and support across the enterprise.

The diagram below illustrates the options available for Level 1 upgrades.

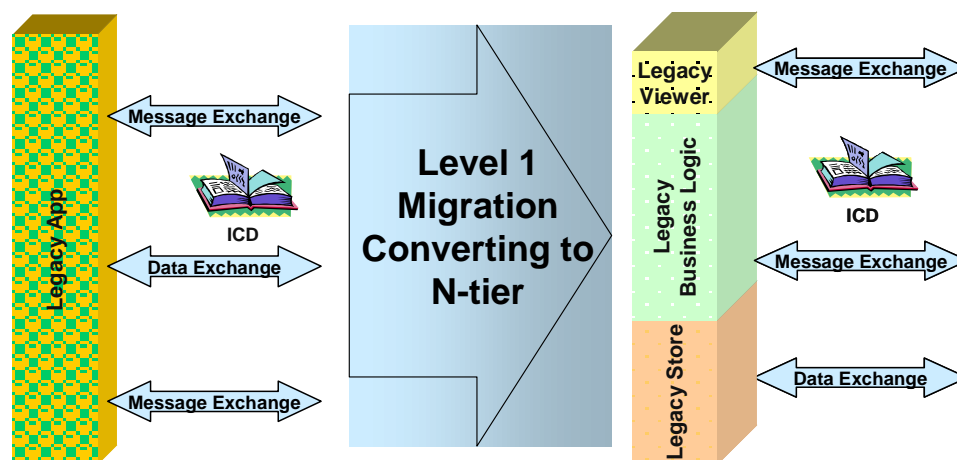


Figure 2 – Level 1 Migration: Conversion to N-Tier

4.3.1 Overarching

1. Identify additional development efforts for the specific environment, such as IT-21, GCCS, NMCI, and so on.
2. Develop and publish JUnit⁷ or automated tests, depending on the implementation language, for all public APIs.⁸
3. Assess the level of effort required to refactor the code into at least four tiers: client tier, presentation tier, middle tier, and data tier.
4. Migrate any operating-system-specific support to an abstraction layer and/or use POSIX⁹-compliant operating system (OS) APIs and test on currently supported operating systems and versions.
5. Develop independent version sequences for the application and the public application interfaces so that they can vary separately.

⁷ <http://www.junit.org>

⁸ Automated test drivers are available for various languages.

⁹ Portable Operating System Interface.

6. Create a configuration file helper class, the mechanisms to interact with the configuration file, and the configuration file. (This would be deployment descriptors for Web-based applications or Java EE¹⁰ applications, or a name/value-pair plain text configuration file for other languages. Microsoft applications should migrate off the registry.) Do not necessarily move all parameters to the configuration file in this iteration; do so where it is reasonable. This is more of a placeholder mechanism to support subsequent levels.
7. Assess the level of effort required to support enterprise system management. Initially, applications must be able to send state information periodically or on demand, and receive commands. Communication occurs via an asynchronous communication mechanism.¹¹ Produce an enterprise management strategy document.¹²
8. Assess the application for the level of effort required to support availability. Produce an availability strategy document.
9. Assess the application or program for security. Produce a security policy document.
10. Incorporate a strong password scheme. Passwords should be at least eight characters long and contain at least one uppercase character, one numerical character, and one special character.

4.3.2 Client and Presentation Tier

1. Factor the GUI code into separable code that can be migrated to client and presentation tiers.
2. Publish and use the public APIs.
3. Prepare existing APIs to migrate to separate tiers: client, presentation, middle, and data. Refactor existing APIs rather than writing new ones.
4. Decouple the public API from the rest of the application. Use a construct similar to Interfaces in Java, Abstract, Protocol classes in C++, or a design pattern¹³ such as façade, proxy, adapter, or bridge.
5. Comment the API with Javadoc or a tool that produces Javadoc-type output.¹⁴
6. Migrate any code that accesses a collaborating system in the client tier to either the middle tier or the data tier. Wrap that code in a connector construct to isolate the application from the enterprise.
7. Assess the level of effort required to support portals for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal. Use this to plan development efforts in a subsequent upgrade level.
8. Migrate client-side security features to middle and data tiers.
9. In Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application so that it supports service plug-ins.

¹⁰ Java Platform, Enterprise Edition.

¹¹ For Navy programs, contact SSC San Diego, Code 24202 for more information.

¹² For Navy programs, provide the document to the PEO C4I & Space Technical Director.

¹³ *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson, Vlissides, 1995, Addison-Wesley.

¹⁴ Javadoc-type tools for other languages are available on the Internet.

10. Move obvious configuration parameters from this tier to the configuration file.
11. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters. Remove hard-coded IP addresses and URLs.
12. Add pre-condition checks to all public API parameters.¹⁵
13. Develop a plan for Discretionary Access Control (DAC) for each Web-accessible component and for the migration to net-centric access controls such as Role-Based Access Control (RBAC).
14. Migrate all “magic number” constant values to named constants. In the C++ example below, the number 6 is a magic number.

```
//original code
if(numFiles > 6) {
// do something
}
//reworked code
const int MAXFILES = 6;
. . .
if(numFiles > MAXFILES) {
// do something
}
```

4.3.3 Middle Tier

1. Publish and use the public APIs of the middle tier. The client tier uses this API.
2. Comment the API with Javadoc or a tool that produces Javadoc-type output.
3. Decouple the public API from the rest of the application, using a construct similar to Interfaces in Java or Abstract or Protocol classes in C++. This enables the implementation interface to vary independently from the part of the interface that is visible to the client.
4. Migrate any code that accesses a collaborating system in the middle tier to a connector construct. This isolates the application from the enterprise.
5. Develop connectors to the Directory NCES service through an appropriate design pattern. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
6. Use Secure Sockets Layer (SSL) encryption to pass authentication information.
7. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.
8. Migrate all “magic number” constants to named constants.
9. Add pre-condition checks in all public APIs.

¹⁵ An example is the use of Assert() in C++.

10. Move configuration parameters from this tier to the configuration file.
11. Migrate client-side security features to the middle tier.

4.3.4 Data Tier

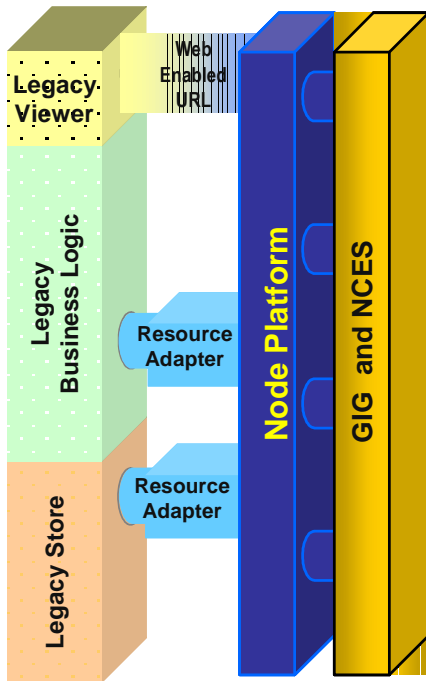
1. Publish and use public APIs. The middle tier uses this API.
2. Comment the API with Javadoc or a tool that produces Javadoc-type output.
3. Migrate any code that accesses a collaborating system in the data tier to a wrapper or connector construct. This isolates the application from the enterprise.
4. Migrate client-side security features to the data tier.
5. Develop connectors to the Directory NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
6. Use Secure Sockets Layer (SSL) encryption to pass authentication information.
7. Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.
8. Migrate all “magic number” constants to named constants.
9. Add pre-condition checks in all public APIs.
10. Move configuration parameters from this tier to the configuration file.

4.4 Level 2: Mid-Level Upgrade

This level contains medium-cost upgrades that enable the application to participate in a net-centric environment at a higher level than Level 1 changes. The theme of Level 2 is to connect factored applications to the Node Platform Infrastructure.¹⁶ The adapters connect the public APIs prepared in Level 1 to the Node Platform interfaces.

The following diagram illustrates the options available for Level 2 upgrades.

¹⁶ See *NESI Part 4: Node Guidance*



Builds on Level 1 Capabilities

- **Do not change legacy viewer, application, or database**
- **Develop node interface to legacy application**
- **Expose data to network via interface to legacy store**
- **Allow users to task legacy application**
- **Allow users to receive status information from legacy application**

Figure 3 – Level 2 Migration: Access to Legacy Data and Applications

4.4.1 Overarching

1. Create a commercial “InstallAnywhere” or “InstallShield” installation script. The script must be executable in multiple runtime environments. If the application resides on a target platform that this class of tools does not currently support, use another approach in the short term. In those cases, devise a migration strategy to open-source installation tools.
2. Incorporate XML-supporting infrastructure and administration.¹⁷
3. Collaborate with XML Namespace Managers to develop an XML representation for the application and COI data. Register this information in the DoD Metadata Repository and Clearinghouse.¹⁸ This may affect the package names for class libraries and naming conventions.
4. Configure the application using an XML-type deployment descriptor model.¹⁹
5. If using an XML parser, develop a wrapper class around it. Code the application to that API to decouple the application from the XML parser.
6. Use validating XML parsers that support the XML schema standard.
7. Develop and publish automated systems integration tests for the entire application.
8. Provide backwards compatibility. Older clients should be able to exchange messages with newer services to execute older functionality.

¹⁷ *Federal XML Developer’s Guide*, http://xml.gov/documents/in_progress/developersguide.pdf.

¹⁸ *DoD Metadata Registry and Clearinghouse*, <http://xml.dod.mil>.

¹⁹ See the Java EE blueprints for more detail on implementation.

9. Provide forward compatibility. Newer clients should be able to exchange messages with older services to execute older functionality.
10. Develop a plan for porting and testing in target operational environments. Include the upgrades necessary to communicate with supporting system applications.
11. Migrate all environment variables to parameterization variables and store them in property files, deployment descriptors, or initialization files. For example, Java EE and Web applications use deployment descriptors.
12. Finish migrating configuration parameters to the external configuration file.
13. Identify proprietary GOTS²⁰ and COTS²¹ code and decouple it via wrapper classes. Design the wrapper classes.
14. Analyze functional areas of the application that will interface to the enterprise.²²
15. Modify application structure to isolate change between the client tier and middle tier, per Level 1 assessment. Enable developers to modify and enhance discrete portions of the enterprise without affecting the others.
16. Develop connectors to the enterprise management service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. At this level, incorporate self-diagnostics, enterprise management reports, and on/off functionality.²³

4.4.2 Client and Presentation Tier

1. Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for Web-based applications.
2. Transform Windows-based applications to be Windows Logo-compliant.
3. Implement COE decoupling components specific to this tier.²⁴
4. Develop portal support for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal, based on earlier assessments. Base migration to the GIG Portal on JSR 168 for Java-based portlets.
5. For Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application in order to support service plug-ins per Level 1 assessments.

4.4.3 Middle Tier

1. Develop connectors to the Messaging NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.

²⁰ Government off-the-shelf.

²¹ Commercial off-the-shelf.

²² Example: include user management and authentication.

²³ Consider Java-based JMX technologies. See <http://java.sun.com/products/JavaManagement>.

²⁴ The high-level COE migration strategy is outlined in section 5, *Migrating COE Systems and Applications*.

2. Develop connectors to the IA/Security enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
3. Develop connectors to the Discovery NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
4. Develop connectors to the Net Time enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
5. Develop connectors to the Network Management enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.
6. Develop connectors to an external directory service for authentication. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. Some application servers have a realm database as part of the application server. This realm database must be “pluggable” to support the use of other directory servers.
7. Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for Web-based applications.
8. For Java applications, develop Discretionary Access Control (DAC) based on container-managed security and the enterprise connector frameworks. Isolate the access control for migration to net-centric Role-Based Access Control (RBAC) authorization services.
9. For C, C++, and ADA applications, develop Discretionary Access Control (DAC) based on a container-managed security model using the Lightweight Directory Access Protocol (LDAP). Isolate the access control for migration to net-centric RBAC authorization services.
10. Migrate from raw sockets and primitive connection APIs to an abstraction layer.
11. Implement container-managed transactions and a concurrency control model.
12. Implement COE decoupling components specific to this tier.
13. Implement application integration and backend integration with initial collaborators.

4.4.4 Data Tier

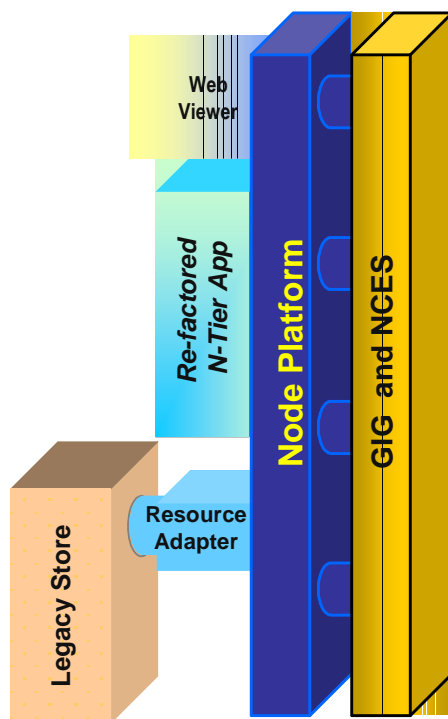
1. Collaborate with Data Area Managers to develop a strategy to incorporate enterprise data policies. Identify, catalog, and report Data Area Manager requirements including data formats, database and versions, authoritative data sources, stored procedures and triggers, and data latency and integrity issues.
2. Coordinate shared resources with collaborators.
3. Implement COE decoupling components specific to this tier.
4. Implement container-managed transactions and a concurrency control model.
5. Create backwardly compatible data mappings for messages.
6. Remove global accounts from databases and integrate them into authentication/access control components.
7. Implement a data integrity scheme for ensuring correct data management when the database is accessed from multiple locations.
8. Implement application integration and backend integration with initial collaborators.

9. Create and publish content metadata in accordance with the guidance from the DoD Metadata Registry and Clearinghouse.²⁵
10. Isolate the application's data tier from the rest of the application with an open-standards CLI interface layer like ODBC, JDBC, a RogueWave-like layer, or an equivalent abstraction.
11. For database applications, migrate from proprietary SQL to ANSI STD SQL 92 or ANSI STD 99, depending on the database. If the application may lose functionality by migrating off proprietary SQL, use an alternative approach. For data stream applications, develop a wrapper abstraction layer that insulates proprietary protocols from the rest of the application.
12. Define the multinational sharing requirements for the data that the service will create and use.

4.5 Level 3: Net-Centric Upgrade

This level contains the higher-cost upgrades that enable the application to participate fully in a net-centric environment. The theme for Level 3 is to migrate the refactored application code to the Node Platform Infrastructure and prepare the application for the net-centric enterprise environment (e.g., namespaces, XML, metadata, publish-subscribe interfaces).

The diagram below illustrates the options available for Level 3 upgrades.



Builds on Level 2 Capabilities

- Rewrite parts (or all) of legacy application as N-tier application (J2EE or .NET)
- Make application part of node workflow
- Access application information via URL or through web portal using a browser
- Allow publish and subscribe with web alerts

Figure 4 – Level 3 Migration: Transform Legacy Applications

²⁵ DoD Metadata Registry and Clearinghouse, <http://metadata.dod.mil>.

4.5.1 Overarching

1. Develop a remote administration/management model that includes frameworks and connectors for remote monitoring, server resource management, and remote software upgrades and maintenance.²⁶
2. Develop remote policies for administrators, operators, and developers.
3. Develop remote installation procedures for components and applications.
4. Implement enterprise authentication and single sign-on, using the connectors developed earlier to facilitate access to data and logic. Use XML-based security assertions to pass authentication information.
5. Integrate application components with the enterprise namespace strategy.
6. Implement return-code-to-exception mapping for applications that use return codes.
7. Place exceptions in the public API descriptions.
8. Develop non-real-time to real-time bridge designs with collaborators.
9. Implement initial implementation of enterprise data policies.
10. Test the application on target operating systems in current and planned operational environments. Include all supporting system applications.
11. Test the application on middleware, including application servers and object request brokers (ORBs), in current and planned operational environments. Include all supporting system application interactions.
12. Develop and publish automated acceptance tests for the entire application.
13. Develop proprietary GOTS and COTS wrapper classes and integrate them into the application.
14. Modify the application structure to isolate change between the middle tier and data tier, per Level 1 assessments. Enable developers to modify and enhance discrete portions of the enterprise without affecting the others.
15. Implement consistent XML data formats, services such as WSDL, and protocols such as SOAP to support data and service exchange across distributed nodes.
16. Assess data for integration with COI or enterprise language and ontologies (e.g., C2IEDM).
17. Implement the enterprise management connector back ends. Integrate with the enterprise management service for on/off, heartbeat, and reports.

4.5.2 Client and Presentation Tier

1. Migrate decoupled clients to Web-page or decoupled-thick-client GUIs. Be able to download them independently from the application. The client module must be able to communicate with the presentation tier on the server via SSL.
2. Restructure the source code to use the enterprise namespace strategy.

²⁶ Software must be able to be installed over the network.

3. Implement application integration and backend integration with more collaborators.
4. Implement return-code-to-exception mapping.
5. Begin implementing enterprise data policies.
6. Develop and publish automated acceptance tests.
7. Develop the enterprise interface connectors and integrate them into the application.

4.5.3 Middle Tier

1. Develop a non-repudiation scheme for application service-to-service interactions.
2. Implement a services-based access model for business logic and data, with support for legacy in/out messages. Exchange node-to-node information through services in the middle tier.
3. Migrate the business logic from the data tier and client tier into the middle tier. Some of the business logic may be contained in the database and not affect the risk strategy.
4. Implement return-code-to-exception mapping.
5. Integrate the application namespace with the enterprise namespace strategy.
6. Implement application integration and backend integration with more collaborators.
7. Begin implementing enterprise data policies.
8. Develop and publish automated acceptance tests.

4.5.4 Data Tier

1. Migrate data tier items from the client and middle tiers into the data tier.
2. Implement return-code-to-exception mapping.
3. Implement a services-based data access model with support for legacy in/out messages.
4. Implement an application collaboration and mediation framework.
5. Incorporate XML-supporting infrastructure and administration.
6. Integrate the application namespace with the enterprise namespace strategy.
7. Implement application integration and backend integration with more collaborators.
8. Begin implementing enterprise data policies.
9. Develop and publish automated acceptance tests.
10. For raw byte-stream data applications and sensors, create an object-oriented wrapper abstraction layer.

4.6 Level 4: Full Net-Centric Integration

Migrating legacy applications to a net-centric environment after Level 3 will require major development efforts.²⁷ The theme of this level is to take a factored, tiered application and provide

²⁷ See *NESI Part 1: Overview* and *NESI Part 2: ASD(NII) Checklist Guidance*

implementation-independent (e.g., XML-based) information exchange, new services for other nodes, and the ability to consume services provided by other nodes.

The diagram below illustrates the options available for upgrades beyond Level 3.

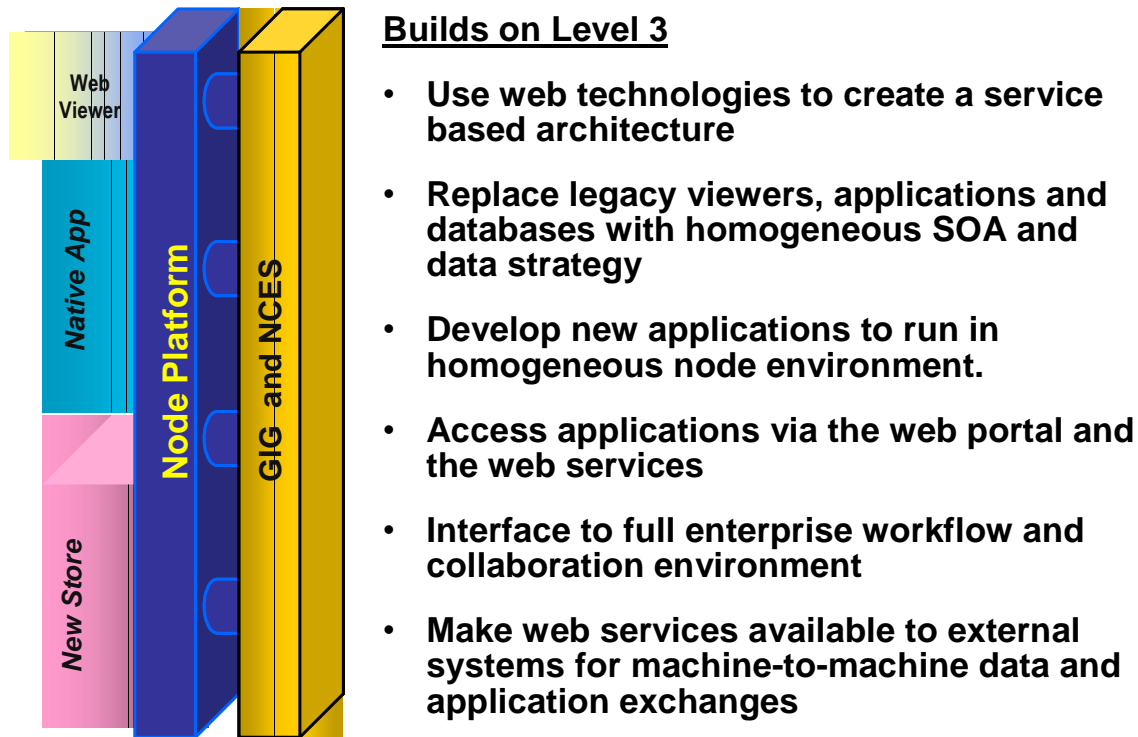


Figure 5 – Level 4: Full Net-Centric Integration

4.6.1 Overarching

1. For applications using RDBMS replication, develop an RDBMS replication strategy to migrate to third-party replication providers and decouple from proprietary replication engines. The replication technology must support replication across vendor databases and versions.
2. Develop an application lifecycle framework that shows how the application interacts with the enterprise.
3. Implement or interface to enterprise caching and communications servers.
4. Implement a cross-domain solution or use CDS enterprise services.
5. Develop a strategy to emulate or gateway legacy networks so that all users are perceived as IP nodes on a larger network.
6. Implement a non-real-time to real-time bridge with collaborators.
7. Implement XML-supporting infrastructure and administration for the enterprise.
8. Implement remote installation of components and applications.
9. Implement availability- and fault-tolerant services.
10. Implement load balancing.

11. Implement database integration.
12. Implement connectors to information management components to include the following:
 - Indexing – content metadata
 - Searching – to parse indexed material
 - Preferences/customization/personalization
 - Profiling
13. Develop interfaces and connectors to Content Delivery Network collaborators.
14. Develop interfaces and connectors to Intelligent Agents.
15. Develop interfaces and connectors to a Profile Management system.
16. Develop interfaces and connectors to a Workflow Management system.
17. Develop interfaces and connectors to a Process Management system.
18. Develop interfaces and connectors to Local Management subsystems.
19. Support strong authentication techniques using centralized authentication servers.

5 Migrating COE Systems and Applications

As the DoD moves toward net-centricity, systems and applications may migrate away from the Common Operating Environment (COE). Migrating COE systems to a net-centric infrastructure requires analyzing the system's dependencies on the COE. Some systems are built to run on the COE without major dependencies; others use complex COE-based functionality. The guidance NESI provides on developing net-centric systems will help bridge the gap.

5.1 Selecting an Approach

COE applications will migrate in phases and at various levels of decoupling. The spectrum of legacy integration and transition possibilities requires multiple integration approaches. A customized approach should mitigate the level of risk and leverage maintenance dollars.

The best approach for a given system depends on two major factors:

- Will the system have different users, requirements, and interfaces than it does now? If so, more flexibility and resources may be needed for the transition.
- What level of COE integration does the system have, how is the implementation achieved, what COE components does the system use, and are there plans for developing non-COE versions of those components?

COE components include workstation and user interface facilities, message processing facilities, and mechanisms for software builds and configuration management.

As the net-centric environment expands, COE functionality will be iteratively replaced by services. Over time, fewer and fewer applications will rely on the COE.

5.2 Analyzing COE Capabilities

The COE provides a number of basic, DISA-provided capabilities, listed below. Use these as the starting point for mapping and transitioning COE capabilities to net-centric, open-standards capabilities.

- An operating system environment
- A GOTS COE kernel that includes the following:
 - A packaging and installation technology, COE segmentation, and the COE Installer
 - Security templates for Discretionary Access Control
 - Account Manager
 - Profile Manager
 - Auditing
 - Process management
 - Platform configuration management

The COE also offers a number of COTS and GOTS products for applications. The migration strategy should consider the number and level of dependencies on GOTS components that are not available outside the COE environment. The major COE GOTS components of interest follow:

- Alerts Services
- Joint Mapping Tool Kit (JMTK)
- Common Message Processor (CMP)
- Integrated C4I System Framework (ICSF)

Each of these products requires a particular migration strategy.

5.3 Decision Tree

Use the following decision tree to determine the best migration option for each COE capability.

1. If your system currently supports multiple environments, and you choose to continue multi-OS (heterogeneous) support, you should provide some form of common interface similar to the COE interface. You can:
 - Build your own version of that capability.
 - Port forward the existing COE implementation.
 - Buy a commercial product to accomplish the same function and migrate your system to that product.
2. If your system currently supports multiple environments, and you choose to support only a single environment, you can:
 - Buy and use the commercial solution set for that environment (e.g., the Navy's NMCI approach).
 - Build an abstraction layer above that environment to isolate your system. Towards this end, you can either:
 - Build your own version of that capability.
 - Port forward the existing COE implementation.
 - Buy a commercial product to accomplish the same function and migrate your system to that product.
3. If your system currently supports multiple environments, and you choose to implement multiple solutions, one per environment, these solutions will not be interoperable. For each environment, you can:
 - Buy and use the commercial solution set for that environment.
 - Build an abstraction layer above that environment to isolate your system. You can either:
 - Build your own version of that capability.
 - Port forward the existing COE implementation.

- Buy a commercial product to accomplish the same function and migrate your system to that product.

5.4 Examples of Mixed COE/Non-COE Systems

For insight into COE migration techniques, examine systems that have developed or are developing mixed COE/non-COE systems:

- I3 integration framework developers toolkit²⁸
- Joint Enterprise DoDIIS Infrastructure (JEDI)²⁹
- eXtensible Tactical C4I Framework (XTCF)³⁰

5.5 Migrating Systems with Basic COE Dependency

Systems with a basic COE dependency only rely on the COE installer and the GOTS COE kernel. The migration strategy in this case is relatively straightforward.

1. Identify the standard COTS components used (e.g., RDBMS) and provide for non-COE versions. These may be provided by the system or by the node on which the system runs.
2. Remove any COE segmentation and COE installer components. Use tools such as MakeInstall and UnMakeInstall to make the system a segment or not a segment, as appropriate.
3. For GOTS, develop an installation procedure using commercial installation technology (e.g., InstallShield for Windows, InstallAnywhere for multiple platforms).
4. Implement Logo compliance on Windows.
5. Implement Appcert compliance on Solaris.
6. Reserve and deconflict machine resources, file system conventions, environment variables, and port numbers.
7. Use only published APIs for the OS.
8. Provide for user and group account management. Set file and directory permissions and password management.
9. Provide process management configuration rules such as what processes run and when. Note that COE-specific process management differs from what the target environment provides.
10. After migration, if the system will provide its own OS and hardware, use the NSA-developed COE security lockdown directions to develop an equivalent security lockdown procedure as part of installation. The general approach is to lock down everything and document those functions that need to be unlocked.

²⁸ Contact the GCCS-M Program Management Office.

²⁹ Contact the JEDI Program Management Office, AFRL/IFEB, <http://extranet.if.af.mil/jedi/> or <mailto:jedi@rl.af.mil>.

³⁰ Contact the XTCF Program Management Office.

11. After migration, if the system will be hosted on another system, develop the security configuration required for proper operation in that environment (e.g., which ports must be open).

5.6 Migrating Systems with COE Component Dependencies

Review the COE components in this section and identify the ones on which the system depends. Develop a specific migration strategy for each one. Each section below makes suggestions that can serve as a starting point. Specific requirements determine the most cost-effective approach.

Some components require multiple migration steps. Others require interim development tactics such as COE to non-COE bridge segments. There may be alternatives to the suggested strategies for some components.

5.6.1 COE Alerts Dependency

- Investigate alternate NCES, OS, or node alert/notification/messaging processing.

5.6.2 COE APM/CDS Dependency

- Provide APM/CDS as a plug-in (subset appropriate to application usage).
- Replace individual invocations with alternate NCES, OS, or node service calls.

5.6.3 COE JMTK³¹ Dependency

- Port any needed JMTK to a non-COE environment. There are several ongoing activities to get off the JMTK.
- Investigate porting the system to use C/JMTK³² or alternate mapping packages such as an OGC-visualization-independent layer.

5.6.4 COE CMP Dependency

- Investigate IRIS.³³
- Investigate a CMP migration strategy.

5.6.5 COE ICSF Dependency

- Include the TMS (CST, etc.), UCP (netproc, etc.), and JMTK subcomponents. This is typically a complex effort.
- Analyze any other ICSF-specific applications that are used.
- Investigate ongoing Web-enabling efforts such as WebCOP.
- Investigate the DISA-sponsored User Defined Operation Picture (UDOP) program.

³¹ See <http://www.jmtk.org> for details on JMTK.

³² See <http://www.cjmtk.com> for details on C/JMTK.

³³ See <http://www.sseusa.com> for details on IRIS.

5.7 Migrating COE Components Using a Bridge Approach

For many COE components, a bridge design pattern³⁴ approach may be appropriate.

In the bridge approach, COE capabilities continue to exist. A new companion segment, called a bridge segment, is installed on the COE system. This segment provides an interface by which net-centric systems can access the COE-based functionality. It also provides net-centric services that make the component functionality available on the network. New, non-COE applications invoke the bridge segment services to access the underlying COE-based functionality.

Depending on the number of public APIs that have to be rewritten, this interim strategy may be more cost-effective than strategies such as wholesale segment conversion. The following diagrams illustrate the bridge approach.

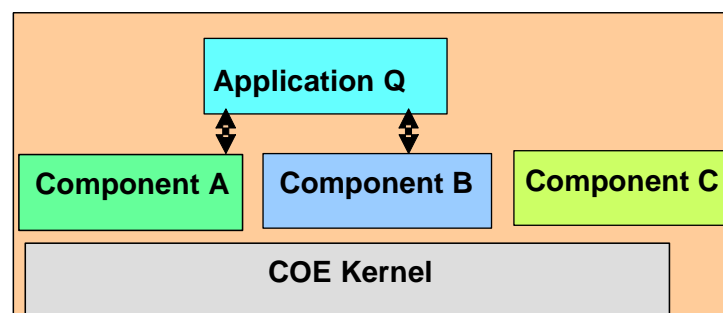


Figure 6 – Notional COE-Based System

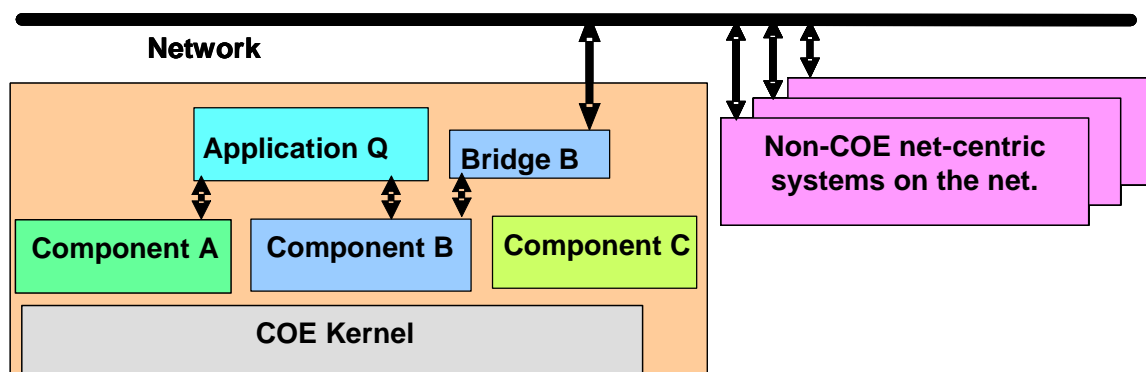


Figure 7 – Notional COE Component Hybrid Bridge Configuration

³⁴ *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson, Vlissides, 1995, Addison-Wesley.

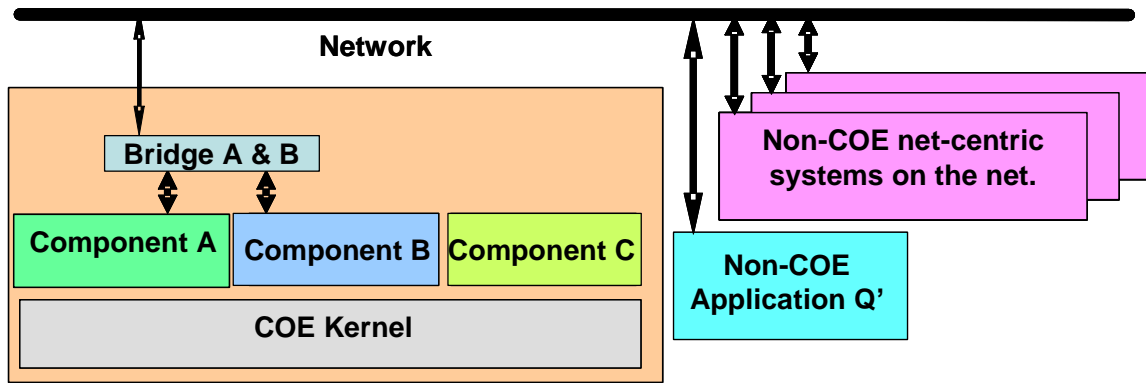


Figure 8 – Notional COE Application Hybrid Bridge Configuration

6 Mapping Maintenance Actions to Enterprise Technology Objectives

This section maps the maintenance actions identified above to the eight enterprise technology objectives included in Table 2, *NESI Part 1: Overview*. The table below illustrates how the levels of integration shown above flow into each other and support the flow among maintenance actions.

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 1 Overarching									
1	Identify additional development efforts for the specific environment, such as IT-21, GCCS, NMCI, and so on.	X	X	X	X	X	X	X	X
2	Develop and publish JUnit or automated tests, depending on the implementation language, for all public APIs.	X	X	X				X	
3	Assess the level of effort required to refactor the code into at least four tiers: client tier, presentation tier, middle tier, and data tier.	X		X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
4	Migrate any operating-system-specific support to an abstraction layer and/or use POSIX-compliant OS APIs and test on currently supported operating systems and versions.	X	X	X				X	
5	Develop independent versioning for the application and the public application interfaces so that they can vary separately.	X	X	X				X	
6	Create a configuration file helper class, the mechanisms to interact with the configuration file, and the configuration file.	X		X		X		X	
7	Assess the level of effort required to support enterprise system management.	X	X			X	X		X
8	Assess the application for the level of effort required to support availability.	X	X		X	X	X		X
9	Assess the application or program for security. Produce a security policy document.	X	X		X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
10	Incorporate a strong password scheme.	X	X		X		X		X
Level 1 Client/Presentation Tier									
1	Factor the GUI code into separable code that can be migrated to client and presentation tiers	X	X					X	
2	Publish and use the public APIs	X	X	X		X		X	X
3	Prepare existing APIs to migrate to separate tiers: client, presentation, middle, and data.	X	X	X		X		X	X
4	Decouple the public API from the rest of the application. Use a construct similar to Interfaces in Java, Abstract or Protocol classes in C++, or a design pattern such as façade, proxy, adapter, or bridge.	X	X	X				X	
5	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Migrate any code that accesses a collaborating system in the client tier to either the middle tier or the data tier. Wrap that code in a connector construct to isolate the application from the enterprise.	X	X	X				X	
7	Assess the level of effort required to support portals for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal. Use this to plan development efforts in a subsequent upgrade level.	X		X	X				
8	Migrate client-side security features to middle and data tiers.	X	X		X		X		X
9	In Motif/X-window applications, implement a design pattern like façade, bridge, or proxy. This decouples the Motif from the rest of the application so that it supports service plug-ins.	X	X	X				X	
10	Move obvious configuration parameters from this tier to the configuration file.	X		X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
11	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters. Remove hard-coded IP addresses and URLs.	X		X		X		X	
12	Add pre-condition checks to all public API parameters.	X	X	X			X	X	X
13	Develop a plan for Discretionary Access Control (DAC) for each Web-accessible component and for the migration to net-centric access controls such as RBAC.	X	X		X		X		X
14	Migrate all “magic number” constant values to constant variables.			X		X		X	
Level 1 Middle Tier									
1	Publish and use the public APIs of the middle tier.	X	X	X				X	
2	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
3	Decouple the public API from the rest of the application, using a construct similar to Interfaces in Java or Abstract or Protocol classes in C++.	X	X	X				X	
4	Migrate any code that accesses a collaborating system in the middle tier to a connector construct.	X	X			X	X		X
5	Develop connectors to the Directory NCES service through an appropriate design pattern. Implement the application side and incorporate a "Not Implemented" exception on the enterprise side.	X	X	X				X	
6	Use SSL encryption to pass authentication information.	X	X		X		X		X
7	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.	X		X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
8	Migrate all “magic number” constants to constant variables.			X		X		X	
9	Add pre-condition checks in all public APIs.	X	X	X			X	X	X
10	Move configuration parameters from this tier to the configuration file.	X		X		X		X	
11	Migrate client-side security features to the middle tier.	X	X		X		X		X
Level 1 Data Tier									
1	Publish and use public APIs.	X	X	X				X	
2	Comment the API with Javadoc or a tool that produces Javadoc-type output.	X	X	X				X	
3	Migrate any code that accesses a collaborating system in the data tier to a wrapper or connector construct.	X	X			X			X
4	Migrate client-side security features to the data tier.	X	X		X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
5	Develop connectors to the Directory NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X						X
6	Use SSL encryption to pass authentication information.	X	X		X		X		X
7	Identify hard-coded constants and parameters that are candidates for external configuration parameters. Migrate these variables to configuration parameters.	X		X		X		X	
8	Migrate all “magic number” constants to constant variables.			X		X		X	
9	Add pre-condition checks in all public APIs.	X	X	X			X	X	X
10	Move configuration parameters from this tier to the configuration file.	X		X		X		X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 2 Overarching									
1	Create a commercial “InstallAnywhere” or “InstallShield” installation script.	X	X					X	
2	Incorporate XML-supporting infrastructure and administration.	X	X	X				X	
3	Collaborate with XML Namespace Managers to develop an XML representation for the application and COI data.	X	X	X				X	
4	Configure the application using an XML-type deployment descriptor model.	X	X	X				X	
5	If using an XML parser, develop a wrapper class around it. Code the application to that API to decouple the application from the XML parser.	X	X	X				X	
6	Use validating XML parsers that support the XML schema standard.	X	X	X			X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
7	Develop and publish automated systems integration tests for the entire application.	X	X	X				X	
8	Provide backwards compatibility.	X	X			X	X		X
9	Provide forward compatibility.	X	X			X	X		X
10	Develop a plan for porting and testing in target operational environments. Include the upgrades necessary to communicate with supporting system applications.	X	X	X				X	
11	Migrate all environment variables to parameterization variables and store them in property files, deployment descriptors, or initialization files.	X	X	X				X	
12	Finish migrating configuration parameters to the external configuration file.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
13	Identify proprietary GOTS and COTS code and decouple it via wrapper classes. Design the wrapper classes.	X	X	X				X	
14	Analyze functional areas of the application that will interface to the enterprise.	X	X				X		X
15	Modify application structure to isolate change between the client tier and middle tier, per Level 1 assessment.	X	X	X				X	
16	Develop connectors to the enterprise management service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. At this level, incorporate self-diagnostics, enterprise management reports, and on/off functionality.	X	X			X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi- User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 2 Client and Presentation Tiers									
1	Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for Web-based applications.	X	X		X	X	X		
2	Transform Windows-based applications to be Windows Logo-compliant.	X	X	X				X	
3	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X
4	Develop portal support for applications migrating to the Navy Enterprise Portal (NEP) or Air Force Portal, based on earlier assessments. Base migration to the GIG Portal on JSR 168 for Java-based portlets.		X		X	X			
5	For Motif/X-window applications, implement a design pattern like façade, bridge, or proxy.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 2 Middle Tier									
1	Develop connectors to the Messaging NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X		X	X	X	X	X
2	Develop connectors to the Audit NCES Services.	X	X	X	X	X	X	X	X
3	Develop connectors to the Discovery NCES service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X	X			X	X	
4	Develop connectors to the Net Time enterprise services. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X				X		X
5	Develop connectors to the Network Management enterprise service. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side.	X	X				X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Develop connectors to an external directory service for authentication. Implement the application side and incorporate a “Not Implemented” exception on the enterprise side. Some application servers have a realm database as part of the application server. This realm database must be “pluggable” to support the use of other directory servers.	X	X		X		X		X
7	Migrate any programmatic security implementations to a container-managed security model. Do not use basic authentication for Web-based applications.	X	X	X		X		X	
8	For Java applications, develop Discretionary Access Control based on container-managed security and the enterprise connector frameworks. Isolate the access control for migration to net-centric RBAC authorization services.	X	X	X	X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
9	For C, C++, and ADA applications, develop Discretionary Access Control based on a container-managed security model using LDAP. Isolate the access control for migration to net-centric RBAC authorization services.	X	X	X	X		X		X
10	Migrate from raw sockets and primitive connection APIs to an abstraction layer.	X	X	X				X	
11	Implement container-managed transactions and a concurrency control model.	X	X		X		X		X
12	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X
13	Implement application integration and backend integration with initial collaborators.	X	X	X					X
Level 2 Data Tier									
1	Collaborate with Data Area Managers to develop a strategy to incorporate enterprise data policies.	X	X				X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
2	Coordinate shared resources with collaborators.	X	X			X	X		X
3	Implement COE decoupling components specific to this tier.	X	X	X			X	X	X
4	Implement container-managed transactions and a concurrency control model.	X	X	X	X	X	X		X
5	Create backwardly compatible data mappings for messages.	X	X	X		X	X		X
6	Remove global accounts from databases and integrate them into authentication/ access control components.	X	X		X	X	X		X
7	Implement a data integrity scheme for ensuring correct data management when the database is accessed from multiple locations.	X	X		X	X	X		X
8	Implement application integration and backend integration with initial collaborators.	X	X	X	X		X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
9	Create and publish content metadata in accordance with the guidance from the <i>DoD Metadata Registry and Clearinghouse</i> .	X	X	X				X	X
10	Isolate the application's data tier from the rest of the application with an open-standards CLI interface layer like ODBC, JDBC, a RogueWave-like layer, or an equivalent abstraction.	X	X	X				X	
11	For database applications, migrate from proprietary SQL to ANSI STD SQL 92 or ANSI STD 99, depending on the database.	X	X	X				X	
12	Define the multinational sharing requirements for the data that the service will create and use.	X	X						

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 3 Overarching									
1	Develop a remote administration/management model that includes frameworks and connectors for remote monitoring, server resource management, and remote software upgrades and maintenance.	X		X				X	
2	Develop remote policies for administrators, operators, and developers.	X	X		X		X	X	X
3	Develop remote installation procedures for components and applications.	X	X	X				X	
4	Implement enterprise authentication and single sign-on, using the connectors developed earlier to facilitate access to data and logic. Use XML-based security assertions to pass authentication information.	X	X		X	X	X		X
5	Integrate application components with the enterprise namespace strategy.	X	X	X		X			X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Implement return-code-to-exception mapping for applications that use return codes.	X	X		X	X	X		X
7	Place exceptions in the public API descriptions.			X				X	
8	Develop non-real-time to real-time bridge designs with collaborators.	X	X				X		X
9	Implement initial implementation of enterprise data policies.	X	X			X	X		X
10	Test the application on target operating systems in current and planned operational environments. Include all supporting system applications.	X	X	X					
11	Test the application on middleware, including application servers and ORBs, in your current and planned operational environments. Include all supporting system application interactions.	X	X	X					

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
12	Develop and publish automated acceptance tests for the entire application.	X	X	X					
13	Develop proprietary GOTS and COTS wrapper classes and integrate them into the application.	X	X	X				X	
14	Modify the application structure to isolate change between the middle tier and data tier, per Level 1 assessments.	X	X	X				X	
15	Implement consistent XML data formats, services such as WSDL, and protocols such as SOAP to support data and service exchange across distributed nodes.	X	X			X	X		X
16	Assess data for integration with COI or enterprise language and ontologies (e.g., C2IEDM).	X	X			X	X		X
17	Implement the enterprise management connector back ends. Integrate with the enterprise management service for on/off, heartbeat, and reports.	X	X			X	X		X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
Level 3 Client and Presentation Tiers									
1	Migrate decoupled clients to Web-page or decoupled-thick-client GUIs. Be able to download them independently from the application. The client module must be able to communicate with the presentation tier on the server via SSL.	X	X					X	
2	Restructure the source code to use the enterprise namespace strategy.	X	X	X				X	
3	Implement application integration and backend integration with more collaborators.	X	X			X		X	
4	Implement return-code-to-exception mapping.	X	X		X	X	X		X
5	Begin implementing enterprise data policies.	X	X			X	X		X
6	Develop and publish automated acceptance tests.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
7	Develop the enterprise interface connectors and integrate them into the application.	X	X		X	X	X		X
Level 3 Middle Tier									
1	Develop a non-repudiation scheme for application service-to-service interactions.	X	X				X		X
2	Implement a services-based access model for business logic and data, with support for legacy in/out messages. Exchange node-to-node information through services in the middle tier.	X	X			X	X		X
3	Migrate the business logic from the data tier and client tier into the middle tier.	X	X	X				X	
4	Implement return-code-to-exception mapping.	X	X		X	X	X		X
5	Integrate the application namespace with the enterprise namespace strategy.	X	X	X				X	

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Implement application integration and backend integration with more collaborators.	X	X			X			X
7	Begin implementing enterprise data policies.	X	X			X	X		X
8	Develop and publish automated acceptance tests.	X	X	X				X	
Level 3 Data Tier									
1	Migrate data tier items from the client and middle tiers into the data tier.			X				X	
2	Implement return-code-to-exception mapping.	X	X		X	X	X		X
3	Implement a services-based data access model with support for legacy in/out messages.	X	X			X	X		X
4	Implement an application collaboration and mediation framework.	X	X		X		X		X
5	Incorporate XML-supporting infrastructure and administration.	X	X	X	X	X	X	X	X

Maintenance Action Options		Capability On Demand	Distributed Operations	Customized Applications	Multi-User Access	Customized Delivery	Assured Sharing	Incremental Upgrade	Data Exchange
6	Integrate the application namespace with the enterprise namespace strategy.	X	X	X		X	X	X	X
7	Implement application integration and backend integration with more collaborators.	X	X			X	X		X
8	Begin implementing enterprise data policies.	X	X				X		X
9	Develop and publish automated acceptance tests.	X	X	X				X	
10	For raw byte-stream data applications and sensors, create an object-oriented wrapper abstraction layer.	X	X			X	X		X